Corrigé détaillé des modules 7 et 8

BTS SIO1 • Deux séances de 60 minutes • Document enseignant

La fiche 7 introduit les listes et leurs parcours. La fiche 8 réutilise cette logique sur les chaînes, puis fait construire un nouveau texte. Les fonctions restent le cadre des productions pour entretenir les acquis de la fiche 6. Les contextes d'inventaire et les exercices sont originaux.

Organisation

Chaque fiche comporte 4 pages : deux pages d'activités, une page de mémo et d'aides, puis un défi facultatif. Prévoir un éditeur Python 3 ; sans poste, faire les traces et les prévisions sur papier. Les productions longues peuvent être écrites au dos. Les défis occupent le temps restant des plus rapides ou une reprise ultérieure ; ils ne s'ajoutent pas aux 60 minutes du parcours.

Étape	Durée	Temps cumulé
Comprendre	12 min	0 à 12 min
Compléter	10 min	12 à 22 min
Traduire	10 min	22 à 32 min
Déboguer	10 min	32 à 42 min
Créer	15 min	42 à 57 min
Bilan individuel	3 min	57 à 60 min

Priorités pédagogiques

Distinguer valeur et indice ; faire compter les éléments avant de donner le dernier indice. Proposer d'abord le parcours direct pour compter ou rechercher, puis le parcours par indices lorsque la position est nécessaire. En fiche 8, comparer précisément les modifications possibles sur une liste et la construction d'une nouvelle chaîne.

Introduire les booléens explicitement : Vrai / Faux en pseudo-code et True / False en Python. Pour une recherche, un succès peut être annoncé dès que l'on trouve la cible ; un échec se conclut après le parcours complet. La liste vide est un test utile et ne nécessite aucun indice.

Aides et observation

Demander un premier essai, puis l'aide 1, l'aide 2 et enfin le pseudo-code complet. Après l'aide complète, proposer une donnée différente et demander une justification. Noter séparément : compréhension du parcours, choix des indices, placement du return et capacité à tester.

Si le temps manque, corriger le débogage ensemble et reprendre la création au début du cours suivant. Ne pas exiger la recherche par indices pour une activité qui porte seulement sur les valeurs. Les raccourcis de recherche et les méthodes de comptage viendront après compréhension de la boucle.

Suite de la progression

Une fiche ultérieure pourra relier chaînes et listes avec split(), int() et la recherche d'un maximum. Les exercices actuels préparent parcours, recherche, filtrage et construction de résultats, sans introduire tirage aléatoire, doublons à éliminer, conversion ou calculs mathématiques supplémentaires.

Module 7 Corrigé des activités

Comprendre les indices

Après l'instruction	stocks	Longueur
Création	[4, 0, 6]	3
stocks[1] = 2	[4, 2, 6]	3
stocks.append(5)	[4, 2, 6, 5]	4

Valeurs finales : stocks[0] vaut 4 ; stocks[3] vaut 5. stocks[4] est invalide : les indices vont de 0 à 3 pour quatre éléments. Une telle lecture lève IndexError en Python. La modification remplace une valeur ; append ajoute un élément sans remplacer les précédents.

Compléter et traduire

Trous : stock = 0 ; nombre + 1 ; nombre. Pour [4, 0, 6, 0], le compteur vaut successivement 0, 1, 1, 2. Il est initialisé avant la boucle pour conserver le nombre déjà trouvé.

```
Fonction compter_vides(stocks : liste) : entier
    nombre ← 0
    Pour Chaque stock dans stocks Faire
        Si stock = 0 Alors
            nombre ← nombre + 1
        Fin Si
    Fin Pour
    Retourner nombre
Fin Fonction

Python
def compter_vides(stocks):
    nombre = 0
    for stock in stocks:
        if stock == 0:
            nombre = nombre + 1
    return nombre

print(compter_vides([4, 0, 6, 0]))
```

Affichage : 2. La variable stock reçoit successivement chaque valeur ; ce n'est pas un indice. Pour une liste sans zéro ou une liste vide, le résultat est 0. return nombre est placé après la boucle pour terminer le comptage avant de retourner le résultat.

Déboguer

Indices tentés : 1, 2, 3. Le programme affiche 0 puis 6, ignore l'élément d'indice 0 (valeur 4), puis échoue sur stocks[3]. len(stocks) est un nombre d'éléments, pas le dernier indice.

```
stocks = [4, 0, 6]
for i in range(len(stocks)):
    print(stocks[i])
```

Sorties corrigées : 4, 0, 6, une par ligne. Accepter aussi for stock in stocks: puis print(stock), car la consigne demande les valeurs et non les indices. Bilan : premier indice 0 ; dernier indice 3 pour quatre éléments ; append ; zéro élément dans [].

Module 7 Recherche et défi

Créer une fonction de recherche

```
Fonction contient(valeurs : liste, cible : entier) : booléen
    Pour Chaque valeur dans valeurs Faire
        Si valeur = cible Alors
            Retourner Vrai
        Fin Si
    Fin Pour
    Retourner Faux
Fin Fonction

Python
def contient(valeurs, cible):
    for valeur in valeurs:
        if valeur == cible:
            return True
    return False
```

Tests : contient([4, 0, 6], 6) → True ; contient([4, 0, 6], 2) → False ; contient([], 6) → False. Le return True peut interrompre le parcours après une correspondance. Le return False est hors de la boucle : le placer dans le else d'un test ferait conclure trop tôt sur une première valeur différente.

Défi de réapprovisionnement

Trous : [] ; stocks[i] < 3 ; i + 1 ; numeros. numeros accumule les numéros retenus. Dans [0, 5, 2, 3], les indices 0 et 2 ont un stock inférieur à 3 : leurs numéros sont 1 et 3.

```
Fonction a_commander(stocks : liste) : liste
    numeros ← []
    Pour i de 0 à Longueur(stocks) - 1 Faire
        Si stocks[i] < 3 Alors
            Ajouter i + 1 à numeros
        Fin Si
    Fin Pour
    Retourner numeros
Fin Fonction

Python
def a_commander(stocks):
    numeros = []
    for i in range(len(stocks)):
        if stocks[i] < 3:
            numeros.append(i + 1)
    return numeros
```

Liste stocks	Liste retournée
[0, 5, 2, 3]	[1, 3]
[3, 4]	[]
[]	[]

Le stock égal à 3 est exclu puisque la consigne dit strictement inférieur à 3. Pour une liste vide, range(0) n'effectue aucun passage et la liste résultat reste vide. La fonction ne modifie pas stocks. Elle retourne une nouvelle liste de numéros, pas une copie des stocks sélectionnés.

Après l'aide, proposer [2, 2, 5] : résultat [1, 2]. Deux stocks identiques correspondent ici à deux matériels distincts ; ils doivent tous les deux être retenus. L'objectif est le filtrage selon un critère, pas la suppression des doublons.

Module 8 Corrigé des activités

Comprendre les caractères

Indice	0	1	2	3
Caractère	P	C	-	4

Sorties : 4 ; P ; -. Le caractère "4" est du texte dans cette référence. reference[4] est hors limites et lève IndexError. Les guillemets ne sont pas comptés. Un espace est compté ; une chaîne vide a une longueur de 0.

Compléter et traduire

Trous : caractere = "-" ; nombre + 1 ; nombre. Pour P, C, -, 4, -, A, les valeurs du compteur sont 0, 0, 1, 1, 2, 2. Résultats : "PC-4-A" → 2 ; "PC4" → 0 ; "" → 0.

```
Fonction compter_tirets(texte : chaîne) : entier
    nombre ← 0
    Pour Chaque caractere dans texte Faire
        Si caractere = "-" Alors
            nombre ← nombre + 1
        Fin Si
    Fin Pour
    Retourner nombre
Fin Fonction

Python
def compter_tirets(texte):
    nombre = 0
    for caractere in texte:
        if caractere == "-":
            nombre = nombre + 1
    return nombre

print(compter_tirets("PC-4-A"))
```

L'appel affiche 2. Le test porte sur le caractère lu, pas sur son indice. Le compteur est initialisé une seule fois et le return se trouve après le parcours.

Déboguer en construisant un nouveau texte

```
reference = "PC-4"
resultat = ""
for caractere in reference:
    if caractere == "-":
        resultat = resultat + "_"
    else:
        resultat = resultat + caractere
print(resultat)
```

Trous : resultat + "_" ; resultat + caractere. Affichage : PC_4. reference conserve "PC-4" et resultat contient "PC_4". L'affectation reference[2] = "_" est impossible sur une chaîne : elle provoque TypeError. On peut en revanche réaffecter une variable à une nouvelle chaîne.

Module 8 Création et défi en pseudo-code

Créer la fonction compter_a

```
Pseudo-code
Fonction compter_a(texte : chaîne) : entier
    nombre ← 0
    Pour Chaque caractere dans texte Faire
        Si caractere = "A" Alors
            nombre ← nombre + 1
        Fin Si
    Fin Pour
    Retourner nombre
Fin Fonction

Python
def compter_a(texte):
    nombre = 0
    for caractere in texte:
        if caractere == "A":
            nombre = nombre + 1
    return nombre
```

Tests : "A-BA" → 2 ; "PC4" → 0 ; "" → 0. Ajouter "AaA" pour vérifier que seul A majuscule est retenu : résultat 2. La fonction retourne un entier, elle ne fait pas d'affichage. Bilan : len("A-B") vaut 3 ; "A" + "B" donne "AB" ; on ne peut pas affecter texte[0] = "X" dans une chaîne.

Défi de nettoyage

Trous : "" ; caractere != "-" ; resultat + caractere ; resultat. Le résultat est une chaîne vide au départ ; chaque caractère différent de "-" est conservé à la fin du résultat. Un tiret ne déclenche aucun ajout.

```
Fonction sans_tirets(texte : chaîne) : chaîne
    resultat ← ""
    Pour Chaque caractere dans texte Faire
        Si caractere != "-" Alors
            resultat ← resultat + caractere
        Fin Si
    Fin Pour
    Retourner resultat
Fin Fonction

propre ← sans_tirets("PC-4-A")
Afficher propre
Afficher Longueur(propre)
```

La fonction conserve l'ordre et les répétitions des autres caractères. Elle ne retire pas les espaces et ne modifie pas le texte reçu. Sur "---", aucun caractère n'est conservé : le résultat est la chaîne vide, pas un espace.

Relances utiles

Si seul le dernier caractère subsiste, vérifier que resultat + caractere est utilisé au lieu de caractere seul. Si un tiret reste, vérifier le test !=. Si un caractère disparaît à tort, faire une trace avec une référence courte contenant deux tirets consécutifs.

Module 8 Traduction et vérification du défi

Programme Python complet

```
def sans_tirets(texte):
    resultat = ""
    for caractere in texte:
        if caractere != "-":
            resultat = resultat + caractere
    return resultat

propre = sans_tirets("PC-4-A")
print(propre)
print(len(propre))
```

Texte fourni	Texte retourné	Longueur
"PC-4-A"	"PC4A"	4
"PC4"	"PC4"	3
"---"	""	0
""	""	0

Le programme principal affiche PC4A puis 4. Une chaîne vide imprimée apparaît comme une ligne vide ; pour inspecter le résultat, on peut temporairement afficher print("[" + propre + "]") afin de rendre les limites visibles.

Trace de la construction du résultat

Caractère lu	Conservé	resultat après ce passage
P	Oui	"P"
C	Oui	"PC"
-	Non	"PC"
4	Oui	"PC4"
-	Non	"PC4"
A	Oui	"PC4A"

Pour sans_tirets(""), la boucle ne passe pas ; resultat reste "" et la fonction le retourne. Dans ces exemples, la longueur retournée correspond au nombre de caractères conservés. Il n'est pas nécessaire d'ajouter un second compteur, puisque len(propre) donne cette longueur.

Observer avant de poursuivre

Vérifier que l'élève sait expliquer pourquoi une liste peut être modifiée par indice et pourquoi la chaîne est reconstruite. Une boucle correcte sur la chaîne vide, un return bien placé et une prévision juste sur "A--B" (résultat "AB", longueur 2) sont de bons indices de compréhension.

Pour la suite, introduire séparément le découpage d'un texte en éléments et la conversion de ces éléments en nombres. Le maximum d'une liste de chaînes ne doit pas être confondu avec celui des entiers qu'elles représentent : ce sera un objectif de la fiche de liaison.